GNU Octave

DMK

Marco Schmid

2017

Inhalte

Inhalte

Was ist GNU Octave?

Interaktive Skriptsprache zur Lösung von Problemstellungen aus dem Bereich der numerischen Mathematik, d.h.

Was ist GNU Octave?

Interaktive Skriptsprache zur Lösung von Problemstellungen aus dem Bereich der numerischen Mathematik, d.h.

- Allgemeine Berechnungen mit Vektoren und Matrizen
- Matrixoperationen, Bildung der Inverse, Zerlegungen, Eigenwerte etc.
- Lösung linearer Gleichungssysteme
- Numerische Lösungsverfahren für so ziemlich alle Bereiche (Gleichungen, ODE etc)
- Ploten (2D, 3D, Vektorfelder, etc)

Matlab:

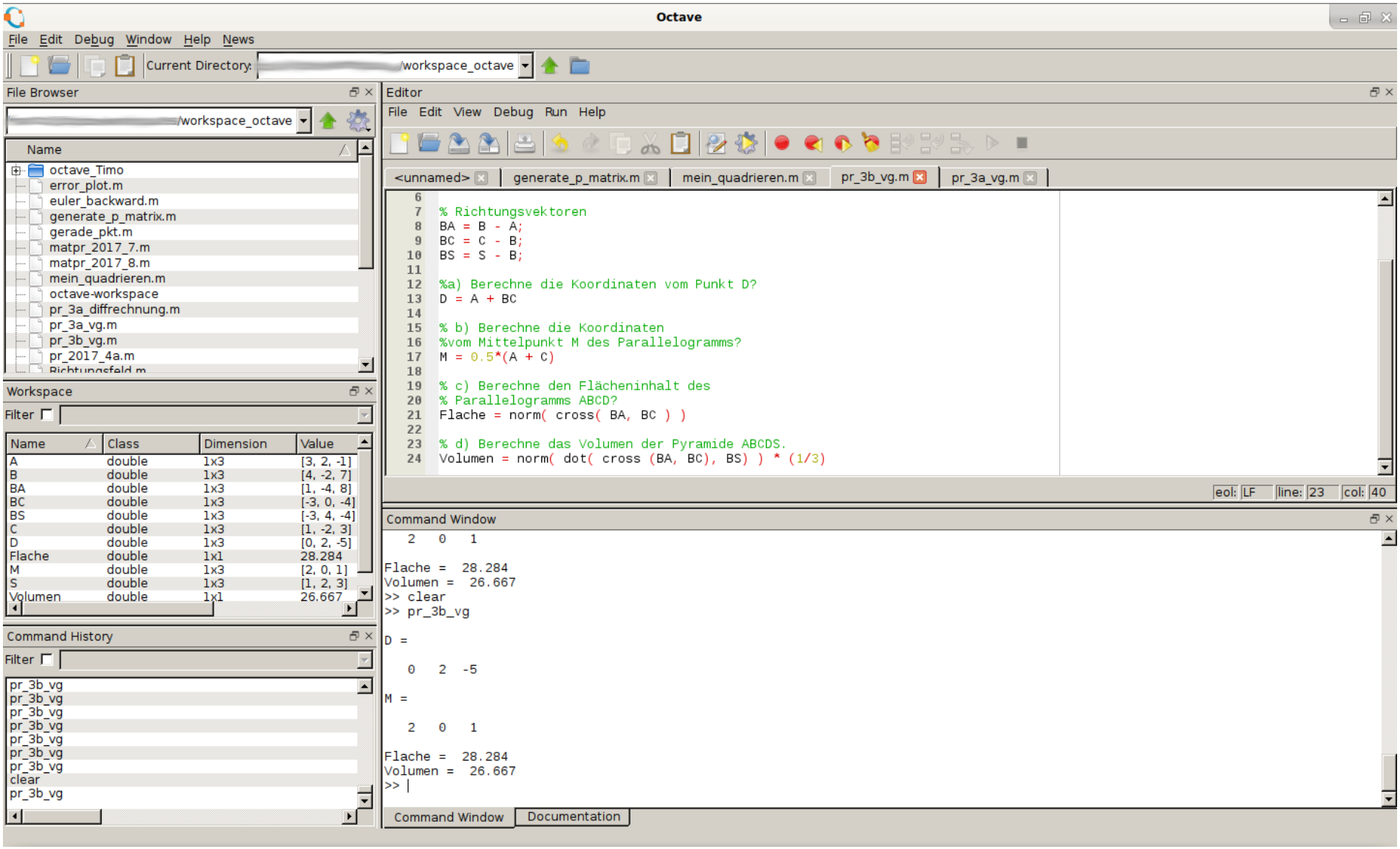
- Syntax von Octave weitgehend mit Matlab identisch
- Ziemlich alle Standardfunktionen deckungsgleich mit Matlab
- Bei Zusatzpaketen kann es etwas anders aussehen

OS

Auf welchen Systemen läuft es?

- Auf allen OS verfügbar
  `https://www.gnu.org/software/octave/`
- Online `https://octave-online.net/`
  (ich habe noch keine Erfahrungen damit)

Oberfläche


Octave
File Edit Debug Window Help News
Current Directory: /workspace_octave
File Browser
/workspace_octave
Name
octave_Timo
error_plot.m
euler_backward.m
generate_p_matrix.m
gerade_pkt.m
matpr_2017_7.m
matpr_2017_8.m
mein_quadrieren.m
octave-workspace
pr_3a_diffrechnung.m
pr_3a_vg.m
pr_3b_vg.m
pr_2017_4a.m
Workspace
Filter
Name Class Dimension Value
A double 1x3 [3, 2, -1]
B double 1x3 [4, -2, 7]
BA double 1x3 [1, -4, 8]
BC double 1x3 [-3, 0, -4]
BS double 1x3 [-3, 4, -4]
C double 1x3 [1, -2, 3]
D double 1x3 [0, 2, -5]
Flache double 1x1 28.284
M double 1x3 [2, 0, 1]
S double 1x3 [1, 2, 3]
Volumen double 1x1 26.667
Command History
Filter
pr_3b_vg
pr_3b_vg
pr_3b_vg
pr_3b_vg
pr_3b_vg
pr_3b_vg
pr_3b_vg
clear
pr_3b_vg
Editor
File Edit View Debug Run Help
<unnamed> generate_p_matrix.m mein_quadrieren.m pr_3b_vg.m pr_3a_vg.m
6
7 % Richtungsvektoren
8 BA = B - A;
9 BC = C - B;
10 BS = S - B;
11
12 %a) Berechne die Koordinaten vom Punkt D?
13 D = A + BC
14
15 % b) Berechne die Koordinaten
16 %vom Mittelpunkt M des Parallelogramms?
17 M = 0.5*(A + C)
18
19 % c) Berechne den Flächeninhalt des
20 % Parallelogramms ABCD?
21 Flache = norm(cross(BA, BC))
22
23 % d) Berechne das Volumen der Pyramide ABCDS.
24 Volumen = norm(dot(cross (BA, BC), BS)) * (1/3)
eol: LF line: 23 col: 40
Command Window
2 0 1
Flache = 28.284
Volumen = 26.667
>> clear
>> pr_3b_vg
D =
0 2 -5
M =
2 0 1
Flache = 28.284
Volumen = 26.667
>>
Command Window Documentation

Inhalte

Vektoren: Definieren

```
>> v = [1,2,3]
v =

1   2   3

>> w = [1;2;3]
w =

1
2
3

>> w'
ans =

1   2   3
>> z = 1:1:3
z =

1   2   3
```

Vektoren: Operationen, Vektor- und Skalarprodukt

File: `vec.m`

```
v = [1,2,3];
w = [1,0,3];

v+w
% Skalarprodukt
v * w' %oder
dot(v,w)

% Vektorprodukt
Kreuz = cross(v,w)
```

Vektoren: Operationen, Vektor- und Skalarprodukt

File: vec.m

```
v = [1,2,3];
w = [1,0,3];

v+w
% Skalarprodukt
v * w' %oder
dot(v,w)

% Vektorprodukt
Kreuz = cross(v,w)
```

Ausgabe:

```
>> vec

ans =

2   2   6

ans =  10
ans =  10
Kreuz =

6   0  -2
```

Beispielaufgabe aus der Vektorgeometrie

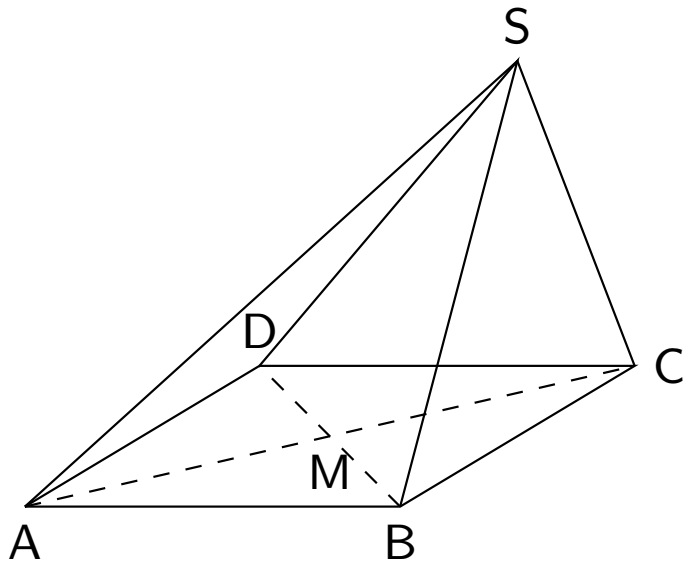
Von einer Pyramide $ABCDS$ mit einem Parallelogramm $ABCD$ als Grundfläche kennt man die Koordinaten der folgenden Punkte: $A(3|2|-1)$, $B(4|-2|7)$, $C(1|-2|3)$ und $S(1|2|3)$

a) Berechne die Koordinaten vom Punkt D?

b) Berechne die Koordinaten vom Mittelpunkt M des Parallelogramms?

c) Berechne den Flächeninhalt des Parallelogramms $ABCD$?

d) Berechne das Volumen der Pyramide $ABCDS$.



File: bsp_vec.m

```
% Ortsvektoren definieren
A = [3,2,-1];
B = [4,-2,7];
C = [1,-2,3];
S = [1,2,3];

% Richtungsvektoren
BA = B - A;
BC = C - B;
BS = S - B;

%a) Berechne die Koordinaten vom Punkt D?
D = A + BC

% b) Berechne die Koordinaten
%vom Mittelpunkt M des Parallelogramms?
M = 0.5*(A + C)

% c) Berechne den Flaecheninhalt des
% Parallelogramms ABCD?
Flache = norm( cross( BA, BC ) )

% d) Berechne das Volumen der Pyramide ABCDS.
Volumen = norm( dot( cross (BA, BC), BS) ) * (1/3)
```

Ausgabe:

```
>> bsp_vec

D =

0    2   -5

M =

2    0    1

Flache =  28.284
Volumen =  26.667
```

Inhalte

Matrizen: Definieren

```
>> A = [1,2,3;4,5,6;0,0,1]
A =

   1   2   3
   4   5   6
   0   0   1

>> B = magic(3)
B =

   8   1   6
   3   5   7
   4   9   2

>> A + B
ans =

   9   3   9
   7  10  13
   4   9   3

>> B * A
ans =

  12  21  36
  23  31  46
  40  53  68
```

Matrizen: Zugriffe

```
>> A = [1,2,3;4,5,6;0,0,1]
A =

1    2    3
4    5    6
0    0    1

>> A(2,2)
ans =  5
>> A(1,:)
ans =

1    2    3

>> A(:,3)
ans =

3
6
1

>> A(1,2:3)
ans =

2    3
```

Matrizen: Determinante, Inverse

```
>> A = [1,2,3;4,5,6;0,0,1]
A =

   1   2   3
   4   5   6
   0   0   1

>> det(A)
ans = -3
>> A^-1
ans =

  -1.66667   0.66667   1.00000
   1.33333  -0.33333  -2.00000
   0.00000   0.00000   1.00000

>> A*A^-1
ans =

   1   0   0
   0   1   0
   0   0   1
```

Matrizen: QR-Zerlegung, Eigenwerte, -vektoren

```
>> [Q, R] = qr(A)
Q =

 -0.24254  -0.97014   0.00000
 -0.97014   0.24254   0.00000
 -0.00000   0.00000   1.00000

R =

 -4.12311  -5.33578  -6.54846
  0.00000  -0.72761  -1.45521
  0.00000   0.00000   1.00000

>> [EV, EW] = eig(A)
EV =

 -0.80690  -0.34372   0.00000
  0.59069  -0.93907  -0.83205
  0.00000   0.00000   0.55470

EW =

Diagonal Matrix

 -0.46410         0         0
        0   6.46410         0
        0         0   1.00000
```

Matrizen: Gleichungssysteme

```
>> A = [1,2,3;4,5,6;0,0,1];
>> b = [9;-2;1];
>> A\b
ans =

-15.3333
10.6667
1.0000

>> T = [A,b]
T =

1    2    3    9
4    5    6   -2
0    0    1    1

>> rref(T)
ans =

1.00000   0.00000   0.00000  -15.33333
0.00000   1.00000   0.00000   10.66667
0.00000   0.00000   1.00000    1.00000
```

Inhalte

Funktionen, kein CAS

- Ein klassischer CAS Solver ist in den Standardfunktionen nicht vorhanden.
- Numerischer Solver: `fsolve`
- Octave als CAS: Paket `symbolic` (basiert auf `sympy`)

  (`symbolic`: siehe nächstes Kapitel)

Funktionen: Definieren

Funktionen «inline» definieren:

```
>> F = @(x) x.^2 - 1
F =
@(x) x .^ 2 - 1
>> F = inline('x.^2 - 1','x')
F = f(x) = x.^2 - 1
>> F(3)
ans =  8
>> F(x)
error: 'x' undefined near line 1 column 3
error: evaluating argument list element number 1
```

Warum ein Punkt?

Funktionen: Definieren

Funktionen «inline» definieren:

```
>> F = @(x) x.^2 - 1
F =
@(x) x .^ 2 - 1
>> F = inline('x.^2 - 1','x')
F = f(x) = x.^2 - 1
>> F(3)
ans =  8
>> F(x)
error: 'x' undefined near line 1 column 3
error: evaluating argument list element number 1
```

Warum ein Punkt? Komponentenweise ausführen:

```
>> c = [1,2,3];
>> c.^2
ans =

1   4   9

>> c^2
error: for A^b, A must be a square matrix. Use .^ for elementwise power.
>> F([1,2])
ans =

0   3
```

Funktionen: Definieren (nicht «inline»)

- Neues File öffnen
- Struktur muss so aussehen:

```
1 function [Rueckgabevariablen] = funktionsname(x,y,...)
2         Berechnung:
3         Werte einfach in die Rueckgabevariablen speichern.
4         Kein explizites "Return" Statement noetig
5 endfunction
```

- File speichern, Name gleich wie Funktionsname!

Funktionen: Definieren (nicht «inline»)

- Neues File öffnen
- Struktur muss so aussehen:

```
1 function [Rueckgabevariablen] = funktionsname(x,y,...)
2         Berechnung:
3         Werte einfach in die Rueckgabevariablen speichern.
4         Kein explizites "Return" Statement noetig
5 endfunction
```

- File speichern, Name gleich wie Funktionsname!

Gleiche Funktion wie vorher (G.m):

```
function y = G(x)
y = x.^2 - 1;
endfunction
```

Funktion kann nun verwendet werden (Achtung File muss sich am gleichen Ort befinden, wo die Ausführung vorgenommen wird).

```
>> G(3)
ans =  8
>> G([1,2,3])
ans =

0   3   8
```

Funktionen: Gleichungen lösen

Da `fsolve` ein numerischer solver ist, wird er nur immer eine Lösung, falls überhaupt, ausspucken.

$$F(x) = G(x) = x^2 - 1$$

```
>> fsolve(F,3)
ans =  1.0000
>> fsolve(G,3)
error: 'x' undefined near line 2 column 9
error: called from
G at line 2 column 7
error: evaluating argument list element number 1
>> fsolve(@G,3)
ans =  1.0000
>> fsolve(@G,-10)
ans = -1.0000
```

Einfacher Funktionsplot

Es werden einfach Punkte verbunden. File: `easy_plot.m`

```
x = -10:0.1:10;
plot (x, sin (x));
xlabel ("x");
ylabel ("sin (x)");
title ("Simple 2-D Plot");
```

Inhalte

Verwendung von Octave als CAS TR

Wie bereits erwähnt: Kein klassischer CAS Rechner!
Mit dem Package `symbolic` (basierend auf `sympy` von Python) jedoch einfach erweiterbar.

Installation von Zusatzpaketen

Wie sieht der Installationsprozess auf den unterschiedlichen Systemen aus?

- Für Linux/Windows/Mac:

  Packages direkt über die Konsole besorgen und installieren:

  `pkg install -forge package_name`

  - Quelle für Linux: z.T. über die Paketquellen verfügbar

  oder ...

- Pakete herunterladen (`*.tar.gz`): Octave-Forge - Extra packages for GNU Octave

  `https://octave.sourceforge.io/`

- Über die octave-konsole installieren:

  `pkg install package_file_name.tar.gz`

- Achtung auf Dependencies: Bsp: `symbolic`

  Bei Linux, falls die Installation über die Packetquellen erfolgen, werden die Abhängigkeiten selber aufgelöst. Bei Windows empfehle ich **OctSympy** (No dependencies, includes a Python interpreter and SymPy)

Paket Symbolic für Windows: Installationsanleitung

1. Folgende ZIP Datei downlaoden:

   `https://github.com/cbm755/octsympy/releases/download/v2.4.0/symbolic-win-py-bundle-2.4.0.zip`

2. Octave öffnen
3. Im File Browser von Octave zum Ort navigieren, in der sich die heruntergeladene Datei befindet.
4. Im Command Window von Octave folgenden Befehl eingeben:

   ```
   pkg install symbolic-win-py-bundle-2.4.0.zip
   ```

   (das kann eine Weile dauern und auch einige Meldungen ausgeben)
5. Das war es auch schon. Zum Testen, folgenden Befehl im Command Window von Octave eingeben:

   ```
   pkg load symbolic
   ```

6. Wenn nichts ausgegeben wird, scheint es zu funktionieren.

Verwendung von Octave als CAS TR

File: bsp_symbolic.m

```
pkg load symbolic
syms x y a b

% Brueche, Dez.zahlen
zahl = 1 / 3
zahl = sym(1/3)
double(zahl)

% Funktionen definieren
f(x) = x^2 -2*x + 1;
g(x) = x^3 + 3*x^2 + 3*x + 1;
h(x) = f(x) * g(x)
ausmultiplizieren = expand(h(x))
faktorisieren = factor(h(x))

% Gleichung
solve( f(x) == g(x), x)

% Int-, Diffrechnung
int(g(x), x)
latex(int(g(x),x))
area = int(f(x), x, 1, 3)
double(area)

dg(x) = diff(g(x),x)
dg(1)

% Gleichungen mehrere Unbekannte
solve( 3*x == 2*y + 10, y -3*x == 0, x, y)

% Limit
g_durch_f = g (x) / f(x)
limit (g(x) / f(x), x, 3)
limit (g(x) / f(x), x, sym(inf))
limit ((g(x)-x^3) / f(x), x, sym(inf))
```